IMPLEMENTANDO ALGORITMOS DE VISÃO COMPUTACIONAL EM VHDL

Paulo Vinícius de Souza[1], **Reinaldo A. C. Bianchi**
Departamento de Engenharia Elétrica
Faculdade de Engenharia Industrial - FEI
Av. Humberto de A. C. Branco, 3972
09850-901 São Bernardo do Campo – SP
rbianchi@cci.fei.br

RESUMO

Partidas de futebol entre robôs constituem uma atividade que possibilita a realização de experimentos reais para o desenvolvimento e testes de robôs, que apresentam comportamento inteligente e que cooperam entre si para a execução de uma tarefa, formando um time. Este artigo descreve o projeto e a implementação do sistema de visão computacional baseado em hardware reconfigurável tipo FPGA para ser utilizado em um time de robôs. Para tanto, são apresentadas a linguagem de definição de hardware VHDL, a descrição dos algoritmos de visão computacional adotados para tratar os problemas específicos do domínio estudado, bem como o sistema implementado. Finalmente, é realizada uma comparação da eficiência do sistema com implementações em software (linguagem C) dos mesmos algoritmos.

ABSTRACT

This paper describes the design and implementation of a field-programmable gate array device (FPGA) based vision system for a robotic soccer team. It presents the VHDL hardware definition language, a description of the computer vision algorithms used for solving domain specific problems, as well as the implemented system. Finally, is made a comparison of the hardware-implemented against a software-implemented (C language) system's efficiency using the same algorithms.

[1] Bolsista Fundação de Ciências Aplicadas – FCA/FEI

1. INTRODUÇÃO

Atualmente, Inteligência Artificial tem sido apresentada a partir de uma visão integrada (Russell e Norvig, 1995). Tal fato é resultado da compreensão, por parte dos pesquisadores, que IA não deve ser vista como segmentada e permite que se trate os problemas dessa área de pesquisa a partir de diversas abordagens.

Na área de visão computacional, essas mudanças se deram por meio do surgimento de novos paradigmas. Visão deixa de ser um problema contido em si mesmo, para tornar-se um sistema que interage com o ambiente através de percepção e ação. Um exemplo é o da visão propositada (Aloimonos, 1994), que considera a visão dentro de um contexto de tarefas que um agente deve realizar, retirando de seus propósitos as restrições para solucionar o problema.

Seguindo essa tendência, os domínios de aplicação pesquisados também começaram a mudar. Na área dos jogos, clássica em IA, criar programas eficientes para jogos de tabuleiro deixou de ser um objetivo distante: no xadrez, os computadores já conseguem vencer os campeões humanos. Novos domínios fizeram-se necessários.

O futebol de robôs foi proposto por diversos pesquisadores (Kitano et al, 1997; Sanderson, 1997) para criar em IA um novo desafio a longo prazo. O desenvolvimento de times de robôs envolve muito mais que integração de técnicas de IA. Segundo Kraetzchmar et al. (1998), "Dispositivos mecatrônicos, hardware especializado para o controle de sensores e atuadores, teoria de controle, interpretação e fusão sensorial, redes neuronais, computação evolutiva, visão, e sistemas multi-agentes" são exemplos de campos envolvidos nesse desafio.

Este artigo apresenta uma solução para uma das áreas do futebol de robôs que oferece grandes dificuldades: a contrução de sistemas visuais para robôs móveis. A solução proposta é a implementação de algoritmos conhecidos de visão computacional em hardware reconfigurável baseados em FPGAs.

Os sistemas baseados em visão computacional têm conquistado espaço tanto nas universidades quanto na indústria, devido à intensa modernização que os sistemas de automação industriais vem sofrendo nos últimos anos. Entre os fatores que impulsionam essa moderização pode-se incluir a competitividade crescente, a rápida alteração dos produtos oferecidos ao mercado e o avanço tecnológico, entre outros, que visam o aumento da produtividade, da qualidade e da confiabilidade dos produtos. Além da área de manufatura e manipulação de matériais, outras aplicações desses sistemas incluem o trabalho em ambientes perigosos ou insalubres, o processamento de imagens de satélites e a exploração espacial.

A proxima seção descreve a linguagem VHDL. A terceira seção descreve os algoritmos de Visão Computacional estudados e a quarta descreve a implementação realizada, apresentando um exemplo de implementação em VHDL. Na quinta seção são apresentados os resultados obtidos e na última parte as conclusões deste trabalho.

2. FPGAs E VHDL

Field Programmable Gate Arrays – FPGAs (Brown & Rose, 1996) são circuito integrado que podem ter o hardware projetado pelo usuário. Um FPGA contém um grande número grande portas lógicas idênticas que podem ser vistas como componentes pré-definidos e que combinam algumas entradas em uma ou duas saídas de acordo com

uma função Booleana especificada pelo usuário que definiu programa. Por sua vez, cada célula é interconectada por uma matriz de conectores e interruptores também programáveis. A vantagem de usar um FPGA é ele permite se atingir velocidades geralmente só alcançadas em sistemas baseados em hardware em aplicações definidas, via software, pelo usuário.

VHDL (Bhasker, 1995) é a abreviação de VHSIC Hardware Description Language (VHSIC é a abreviação de Very High Speed Integrated Circuits). VHDL é uma linguagem de descrição de hardware que pode ser usada para modelar um sistema digital, desde uma simples porta lógica até um sistema digital completo.

A necessidade de uma linguagem padronizada de descrição de hardware surgiu em 1981 no programa VHSIC. Neste, um grande número de empresas dos Estados Unidos desenvolvia CIs para o departamento de defesa (DoD), cada uma usando uma linguagem de descrição diferente das outras – o que não era produtivo, forçando a adoção de uma linguagem comum. Em 1983, um grupo de três companhias - IBM, Texas Instruments e Intermetrics - assinou um contrato com o DoD para desenvolver uma versão da linguagem.

A versão 7.2 do VHDL foi desenvolvida e apresentada ao público em 1985. Após isso, houve uma crescente necessidade de fazer com que a linguagem se tornasse um padrão da indústria. Conseqüentemente, em 1986, a linguagem foi transferida para o IEEE para padronização.

Depois de avanços substanciais na linguagem, feitos por um grupo de indústrias, universidades e representantes do DoD, a linguagem tornou-se padrão pelo IEEE em dezembro de 1987 (IEEE Std 1076-1987). A linguagem passou a ser reconhecida como um padrão do American National Standards Institute (ANSI).

De acordo com as regras do IEEE, um padrão do IEEE precisa ser reformulado a cada 5 anos para que possa continuar a ser um padrão. Conseqüentemente a linguagem foi atualizada com novos recursos, a sintaxe da maioria das construções foi uniformizada, e muitas ambigüidades presentes na versão de 1987 da linguagem foram resolvidas. A versão da linguagem é conhecida como IEEE Std 1076-1993. O DoD, desde setembro de 1988, exige que todos os seus fornecedores de Circuitos Integrados de Aplicação Específica (ASIC) apresentem descrições VHDL dos seus circuitos.

VHDL permite para o projetista implementar a arquitetura por meio de três métodos diferentes: o método de Fluxo de Dados, o método Comportamental, ou de Behavioral, e o método Estrutural.

No método de Fluxo de Dados o projetista define declarações Booleanas que são atribuídas a sinais. Essas declarações são executadas concorrentemente e são usadas para implementar funções Booleanas simples, que dependem de um ou vários sinais binários.

O método de Behavioral define processos que contêm declarações entre um par de Etiquetas “Process” e “End process”. As declarações dentro de um processo são executadas consecutivamente e trabalham da mesma maneira que qualquer programa escrito em uma linguagem procedimental, como C ou Pascal.

Finalmente, o método Estrutural permite ao projetista descrever as ligações elétricas do circuito e conecta vários “componentes virtuais” dentro do FPGA. É usado para definir a interconexão lógica dos componentes de circuito.

Esses três modos podem ser combinados no chamado “modo misturado”, no qual

o projetista pode implementar parte do sistema como um circuito digital e parte como um algoritmo.

3. SEGMENTAÇÃO DE IMAGENS BASEADA EM VISÃO COMPUTACIONAL

Segundo Ballard e Brown (1982) "a idéia de segmentação tem suas origens nos psicólogos do grupo Gestalt, que estudavam as preferências exibidas por seres humanos no agrupamento ou organização de um conjunto de formas dispostas no campo visual". Ainda segundo estes autores, segmentação de imagens é o termo usado em visão computacional para "o agrupamento de partes de uma imagem genérica em unidades que são homogêneas com respeito a uma ou várias características (ou atributos), que resulta em uma imagem segmentada". (Ballard & Brown, 1982, p. 116).

Existem duas maneiras básicas de realizar a segmentação de uma imagem:

- A análise baseada em similaridades das regiões da imagem. Regiões são definidas normalmente como áreas 2D conectadas, sem superposição (um pixel só pode fazer parte de uma única região).
- A análise baseada em descontinuidades na imagem, que utiliza as variações bruscas nos valores de intensidade dos pixels para particionar uma imagem.

O sistema implementado utiliza um algoritmo do tipo *chain code*, baseado no descrito em (Rillo, A. 1989). Nele, dada uma imagem capturada, de tamanho determinado, este algoritmo inicialmente filtra a imagem utilizando um limiar de cor, gerando uma imagem binária. Depois, encontra os contornos da imagem filtrada e finalmente, a partir da imagem de contornos, este algoritmo encontra todas as regiões da imagem que possuem um contorno fechado.

A seguir são apresentados algoritmos que realizam cada parte deste processo, apresentando antes alguns conceitos sobre captura e padrões de imagens.

3.1 Captura de Imagens

Um programa que capture uma imagem colorida (24 bits), no modelo de cor YUV, de tamanho variável, gera as seguintes imagens (Figura 1).

O modelo de cores YUV (Li, 1998) foi inicialmente usado no padrão PAL para vídeo analógico e atualmente é usado como padrão para video digital, sendo que o JPEG e o MPEG são baseados em um modelo YUV modificado.

Este modelo define a imagem como sendo uma matriz de pontos, onde a cor de cada ponto é definido por 3 bytes: o primeiro define a luminância da imagem (Y), e os segundo (U) e terceiro (V) definem a crominância.

A luminância Y é definida a partir das cores vermelha, verde e azul, onde:

$$Y = 0.299 * \text{Vermelho} + 0.587 * \text{Verde} + 0.114 * \text{Azul}$$

A crominância é definida como a diferença entre uma cor de referência e um branco de referência para uma mesma luminância. Assim os valores de crominância são definidos para o azul e o vermelho.

$$U = \text{Azul} - Y$$

$$V = \text{Vermelho} - Y$$

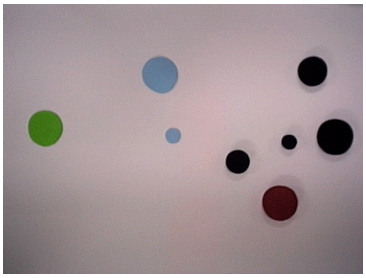
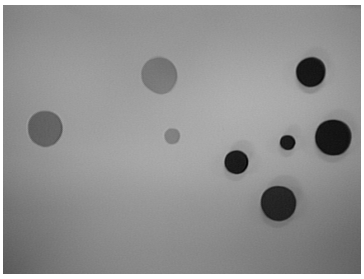
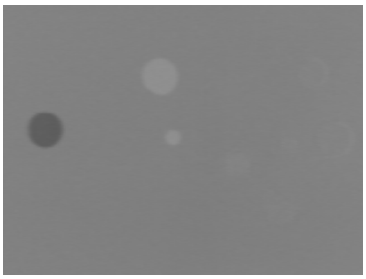


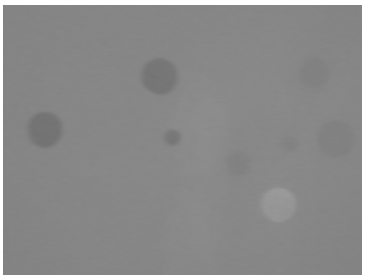
Imagem Composta



Y - Luminância



U - Crominância Azul



V - Crominância Vermelho

Figura 1 - Imagem composta capturada.

Neste modelo de cor uma imagem em níveis de cinza não possui crominância e U e V tornam-se igual a zero. Ainda, V varia do vermelho ao ciano e U do azul ao amarelo.

3.2 Filtragem de Imagens por Cor (colorFilter)

A tarefa primitiva deste algoritmo é filtrar uma imagem colorida, de tamanho variável, construindo uma imagem binária (Figura 2). Ele recebe a imagem colorida e o valor de limiar de uma cor que se deseja filtrar no modelo YUV e atribui 1 às regiões da imagem onde a cor supere o limiar desejado e zero ao resto da imagem.

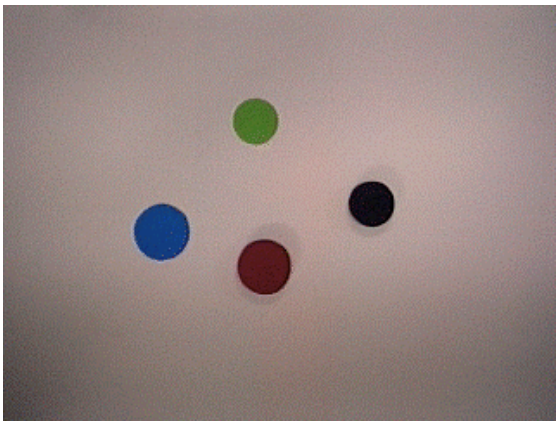


Imagem original

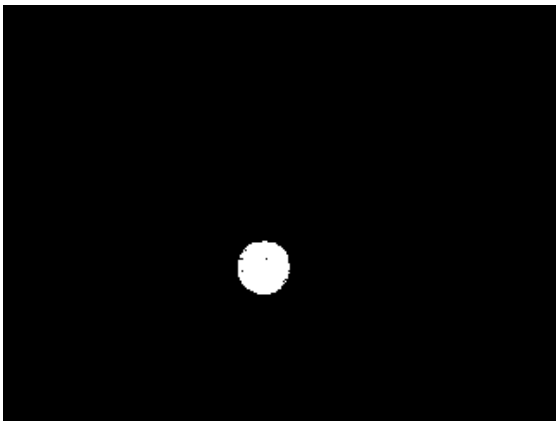


Imagem binarizada

Figura 2 - Imagem binarizada para a cor vermelha.

3.3 Determinação de Contornos (edgeDetector)

Dada uma imagem binária, este agente determina os contornos existentes nela. É baseado no algoritmo descrito em (Rillo, A., 1989), que utiliza a definição de contornos proposta por Kitchin e Pugh (1983). Ele verifica quatro elementos da imagem (máscara 2x2) para determinar se um ponto faz parte de um contorno ou não, criando uma nova imagem onde as bordas encontradas estão entre os elementos da imagem original.

A figura 3 apresenta seis configurações básicas das dezesseis possíveis combinações de quatro elementos de uma imagem binária. As configurações na coluna esquerda não pertencem a regiões de contorno e por isso se atribui zero ao ponto correspondente da imagem de resultante. Os da coluna direita são pontos que pertencem a um contorno e resultam em um. As outras configurações possíveis podem ser obtidas pela rotação destas apresentadas.

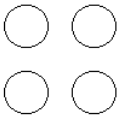
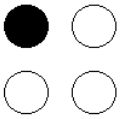
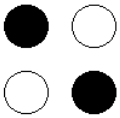
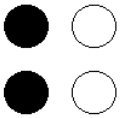
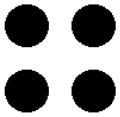
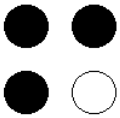
Elementos não pertencentes ao contorno (Resulta 0)	**Elementos pertencentes ao contorno (Resulta 1)**
	
	
	

Figura 3 – Definição dos elementos básicos de borda

Como este processo encontra o contorno entre os elementos da imagem, em vez de dobrar a resolução da imagem resultante - o que seria necessário para desenhar as bordas entre os pixels originais- esta é deslocada de meio pixel para a esquerda e para cima da imagem original, visando manter a resolução original.

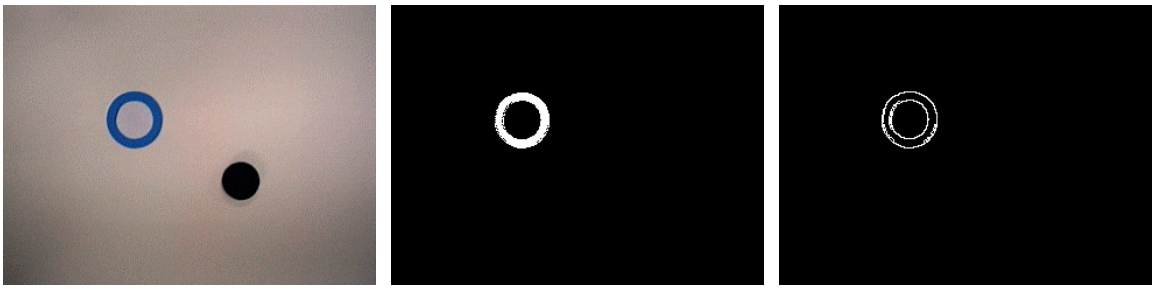


Imagem Capturada | Imagem Filtrada | Imagem dos Contornos

Figura 4 - Determinação de Contornos

A figura 4 mostra uma imagem capturada onde existe um anel azul e uma peça preta, a imagem filtrada para a cor azul e o contorno encontrado.

3.4 Vetorizador (vetorizer)

Dada a imagem binária, de tamanho determinado contendo os contornos de uma imagem, este agente encontra todas as regiões da imagem que possuem um contorno fechado. Ele gera uma lista com a área, o tamanho do contorno mínimo (*chain size*), a posição do centro de área para todas as regiões fechadas da imagem. Ainda, este agente verifica se cada região possui uma forma circular ou não, utilizando a razão entre a área e o contorno calculado.

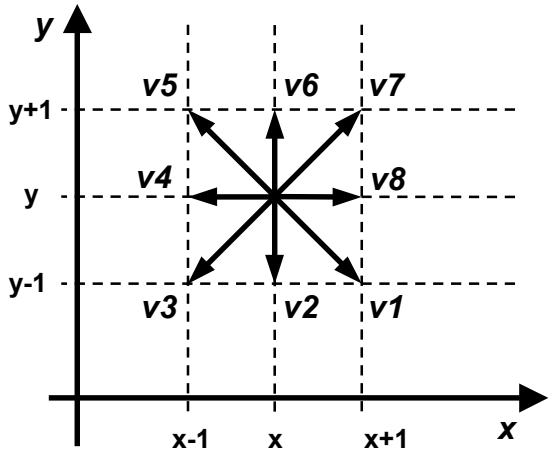


Figura 5 – Definição dos vetores elementares

O agente usa um algoritmo do tipo *chain code* (Ballard & Brown, 1982), baseado no descrito em (Rillo, A., 1989) e em (Kitchin & Pugh, 1983). Este algoritmo considera que qualquer ponto em matriz de imagem está conectado aos oito pontos imediatamente a sua volta através de um vetor elementar, cuja direção foi rotulada (de v1 até v8) e que são apresentados na figura 5.

O contorno de uma região pode ser determinado, assim como sua área calculada, através de um procedimento que segue os contornos de uma imagem, criando uma corrente de vetores ligados. Basta o ponto inicial da corrente e a seqüência dos vetores para se determinar precisamente uma região. Este procedimento segue o contorno mais externo, caso a borda possua mais de um pixel de espessura.

Para calcular a área de uma região é realizado um procedimento semelhante à integração, onde a área total é dada pela somatória da contribuição de cada vetor elementar. A figura 6 mostra as contribuições dos vetores e a tabela 1 apresenta os valores desta contribuição.

O perímetro de uma região pode ser encontrado facilmente através dos algoritmos baseados em *chain-codes*. O perímetro total é:

$$Perímetro = PerimetroPar + PerimetroÍmpar \cdot \sqrt{2}$$

isto é, a soma do número de vetores com rótulo par (que estão na horizontal ou vertical e por isso tem tamanho unitário) mais a soma do vetores com perímetro ímpar (que estão nas diagonais) vezes o tamanho destes, que é $\sqrt{2}$. O centro da região também

pode ser calculado, utilizando fórmulas para o cálculo de centro de área. A figura 7 mostra uma imagem vetorizada.

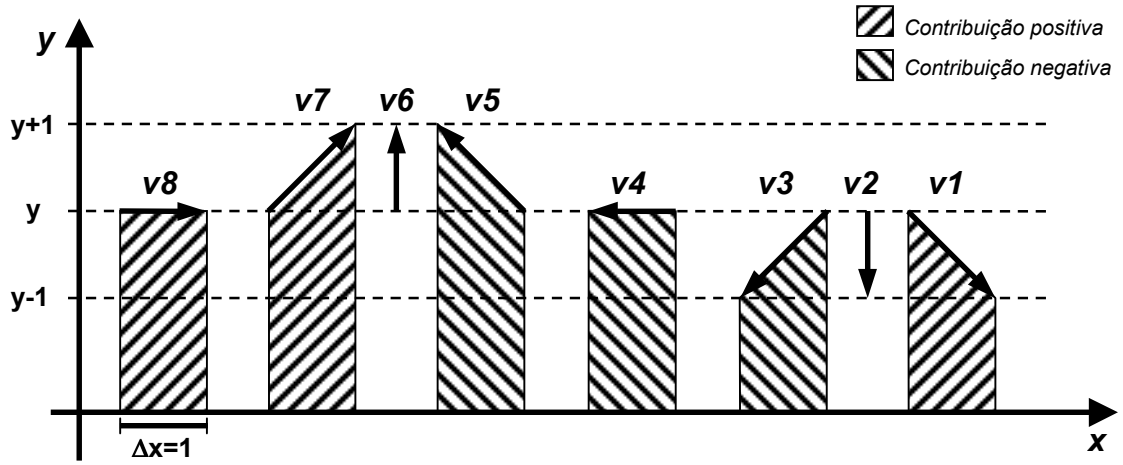


Figura 6 – Contribuição de cada elemento para o cálculo da área

Vetor de Direção	Δ Área
v8	y · 1
v7	(y + ½) · 1
v6	0
v5	(y + ½) · (-1)
v4	y · (-1)
v3	(y - ½) · (-1)
v2	0
v1	(y - ½) · 1

Tabela 1 – Valores da contribuição de área de cada elemento

A figura abaixo apresenta um exemplo de imagem vetorizada.

Imagem Capturada | Imagem Filtrada | Contornos | Imagem Vetorizada

Figura 7 – Imagem vetorizada.

4. O SISTEMA IMPLEMENTADO

O sistema implementado é composto pelas seguintes partes (figura 8):

- Aquisição da Imagem
- Binarização
- Determinação de contornos, com extração das bordas.
- Vetorização, com cálculo de informações sobre a imagem.
- Envio dos dados

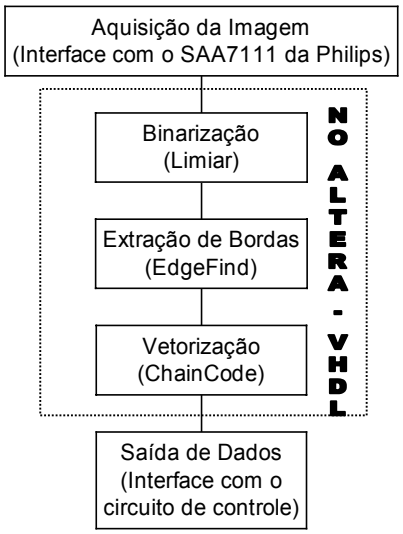


Figura 8 – Diagrama de Blocos do sistema implementado em Hardware.

4.1 Aquisição da imagem

Para a construção do sistema em hardware é necessário encontrar uma forma de extrair imagens diretamente de uma câmera de vídeo. Para isso foi utilizado um decodificador de vídeo da Philips que faz a conversão do sinal da câmera para o padrão RGB, realizando a aquisição da imagem.

O SAA7111 é um CI produzido pela Philips que tem como função receber uma entrada de vídeo analógica e transformá-la em uma saída digital com vários padrões diferentes, à escolha do usuário.

A entrada do processador de vídeo é feita por uma combinação de dois canais analógicos de pré-processamento. Estes canais possuem uma fonte de seleção, um filtro "anti-aliasing", um controle de ganho automático, um circuito gerador de clock (CGC), um decodificador digital multi– padrões (PAL BGHI, PAL M, PAL N, NTSC M e NTSC N), possui um circuito que controla o brilho, contraste, saturação e uma matriz espacial de cores. O SAA7111 aceita como entrada analógica os sinais CVBS ou S-vídeo (Y/C) retirados da TV ou VTR.

A saída do decodificador de vídeo pode ter vários padrões, como por exemplo, YUV (12-bit), YUV (16-bit), YUV (8-bit), RGB (16-bit), RGB (24-bit). Todos estes sinais são digitais.

A imagem utilizada pelo sistema tem resolução de 320 linhas, 240 colunas e 16

bits de cores. Ela é gerada por uma câmera de vídeo comum, capturada e decodificada pelo SAA7111. Neste trabalho foi utilizado como entrada o padrão NTSC M e como saída o padrão RGB de 16 bits.

A interface do decodificador com o FPGA que implementa a visão é feita através de 16 pinos de entrada (RGB16) da seguinte forma: 10 pinos são utilizados para as cores vermelho e azul (5 para cada) e 6 para o verde.

Este circuito é controlado pelo protocolo de vídeo I2c-bus, descrito a seguir.

4.2 O protocolo I²C-bus

O I^2C-bus (Inter Integrated Circuit-bus) é um protocolo de comunicação que foi desenvolvido pela divisão de semicondutores da Philips no início da década de 80. Este protocolo tem por objetivo criar um caminho mais simples para se conectar a CPU de um micro computador a um chip através da comunicação serial bidirecional, composta por dois cabos: a linha serial de dados (SDA) e a linha serial de clock (SDL).

Cada componente conectado é reconhecido pelo software com um único endereço e uma simples relação mestre/escravo é quem faz a comunicação. O mestre pode operar como transmissor ou receptor. Durante a transmissão podemos enviar mensagens sem restrição do número de bytes a serem enviados.

Para podermos configurar o CI utilizando a comunicação serial cridas três sub-rotinas que manipula os sinais SDA e SDL. Essas rotinas são:

- I2C_START = inicia transmissão
- I2C_SEND_BYTE = manda 1 byte
- I2C_STOP = termina transmissão

Para se configurar todos os parâmetros do CI são utilizadas diversas sub-rotinas em seqüência, repetindo o I2C _SEND_BYTE tantas vezes quanto necessário para enviar um bloco completo consistido por:

- Identificador (Device ID) do SAA7111;
- Endereço do primeiro parâmetro a ser programado
- Valor do primeiro parâmetro.
- Valor do parâmetro seguinte.
- Configurações restantes.

4.3 Binarização da Imagem

Após ser capturada, a imagem é convertida para bitmap preto e branco através de um sistema de limiar de cores (como descrito na seção 3.2), que filtra e transfere para a imagem monocromática somente uma cor predefinida. Esta imagem (monocromática) é armazenada em memória para ser processada.

4.4 Determinação dos contornos

A extração de bordas descrita na seção 3.3 é implementada aqui. A imagem é analisada em partes constituídas de quatro pontos cada uma, assim sendo analisa-se primeiro os dois primeiros elementos da primeira linha, juntamente com os dois

primeiros da segunda, depois os segundo e terceiro elementos da primeira e da segunda linha, e assim por diante até o final da linha. Quando termina a linha, repete-se o mesmo processo para a segunda e a terceira linha e assim por diante até que tenha encerrado as linhas.

Durante esta fase, a imagem binaria armazenada anteriormente é seqüencialmente substituída pela imagem contendo as bordas, para diminuir a necessidade do uso de memória.

4.5 Identificação de Objetos na Imagem

A vetorização da imagem (como descrita na seção 3.4) é realizada neste módulo. O resultado do módulo anterior, armazenado na área de memória reservada para a imagem, é passado para a função chaincode que transforma cada borda da imagem em uma seqüência de vetores, a partir dos quais elabora uma matriz com as informações de posição e tamanho de cada objeto. Ele faz uma varredura na imagem gerada pelo primeiro algoritmo, que contém apenas as bordas dos objetos, e cria uma matriz de vetores (chain code) para cada objeto que possua um contorno fechado. O resultado deste algoritmo aplicado à área onde se encontra a bola é mostrado a seguir:

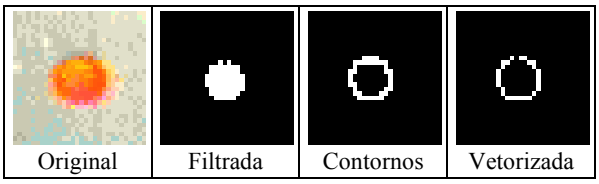


Figura 9 – Resultado de uma imagem Analizada pelo sistema, em suas diversas fases.

Ao final desta análise encontramos as informações sobre cada objeto, que neste caso é toda seqüência de vetores que forma um contorno fechado. Estes dados são enviados para a porta de saída do circuito.

4.6 Saida dos dados

Para realizar a saída dos dados foi elaborado um pacote de dados. Os dados são colocados na saída em pacotes de 32 bits, sendo 9 para a posição do objeto em relação ao eixo X, 9 para o eixo Y e 14 para o tamanho do contorno mínimo (*chain size*). Esta informação pode ser enviada a outro módulo de FPGA, como o módulo de estratégia ou um módulo que transmita a para um computador através de uma porta de comunicação serial RS-232.

4.7 Transposição dos algoritmos para VHDL

Ao implementar o projeto, os três modos para escrever uma especificação de VHDL, descritos na seção 2, foram combinados no chamado “modo-misturado”, onde o projetista pode implementar parte do sistema como um circuito digital e outra parte como um algoritmo. Como os algoritmos implementados no sistema têm um tamanho considerável, nós apresentamos aqui a simplificação do dispositivo de limiarização.

A seguir se encontra um exemplo de uma descrição em VHDL que implementa

um dispositivo de limiarização de uma imagem, no caso um filtro “passa-baixa”. Foi usado o método de fluxo de dados.

```
-- Esta parte descreve os sinais de entrada e saída
Entity Threshold is
   PORT (  R, B:          IN BIT_VECTOR(0 TO 4);
           G:             IN BIT_VECTOR(0 TO 5);
           I:             OUT BIT
 );
End Threshold;

-- Esta parte defina a funcionalidade do dispositivo
Architecture Threshold of threshold Is
BEGIN
   I <= not (R(4) or B(4) or G(5));
End Threshold;
```

A primeira parte do código define os sinais de entrada e saída, onde há 5 bits para os sinais vermelhos e azuis e 6 bits para o sinal verde e apenas um para o sinal de saída. A segunda parte definea função do sistema: um dispositivo que aceitará só cores que possua os componentes vermelho, verde ou azul menos da metade da intensidade máxima. Se algum dos bits mais significativos estiver com valor verdsadeiro, o sinal de saída estará desligado.

5. RESULTADOS

Os algoritmos implementados foram testados em um Pacote de Laboratório do Programa Universitário ALTERA, que inclui Software de desenvolvimento MAX+PLUS II (Edição de Estudante), uma placa de desenvolvimento educacional e um cabo para carga ByteBlaster.

O software MAX+PLUS II é um sistema de desenvolvimento que permite o desenvolvimento através de sistemas gráficos ou de uma linguagem de descrição de hardware, compilação e verificação do projeto, e programação de dispositivo. A placa de desenvolvimento educacional possui dois dispositivos lógicos programáveis e reconfiguráveis. Finalmente, o ByteBlaster permite a carga de programas do MAX+PLUS na placa.

Abaixo podemos ver um trecho da carta de tempo da fase de extração de bordas. O número de clocks gastos para a execução desta fase é o número de linhas da imagem acrescido de um, pois o método de implementação utilizado consegue processar uma linha inteira a cada ciclo. Como o clock utilizado para o processamento da imagem é de 100 MHz e nossa imagem tem 240 linhas, temos um total de 2,41μs para cada imagem.

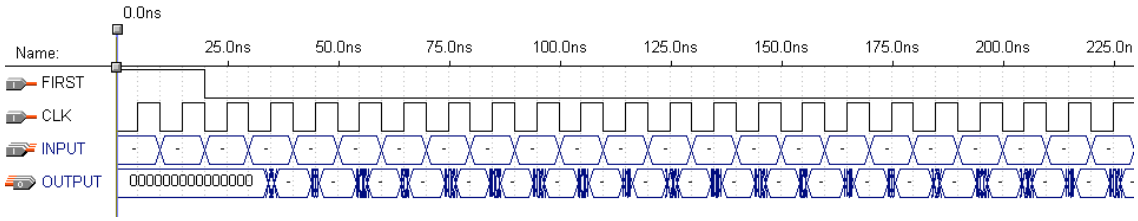


Figura 10 – Carta de tempo.

O sinais na carta de tempo são:

- CLK – clock do sistema;
- INPUT – vetor de entrada que é a representação de uma linha inteira;
- OUTPUT – vetor de saída no mesmo formato da entrada;
- FIRST – sinal de controle de sincronismo;

A tabela abaixo mostra o tempo necessário para o processamento de cada imagem, o que nos mostra que, teoricamente, o sistema poderia processar aproximadamente 649 imagens no formato utilizado por segundo. Os mesmos algoritmos, quando implementados em linguagem C e ambiente Linux, leva 33ms para processar cada imagem, ou seja, 21,5 vezes mais tempo.

Fase	***Tempo***
Captura e Binarização	770µs
Extração de Bordas	2,41µs
Vetorização	770µs
Tempo total até os dados estarem disponíveis na saída	**1,54ms**

Tabela 2 – Tempo necessário para cada módulo.

Assim sendo, quem determinará o número de quadros processados por segundo poderá ser a câmera, o circuito de captura ou o circuito que processará os dados, já que é certo que todos eles terão capacidade inferior à apresentada pela fase de tratamento e identificação de objetos na imagem.

6. CONCLUSÃO

A comparação da eficiência do sistema implementado em hardware com os mesmos algoritmos implementados em linguagem C mostrou que o primeiro sistema alcançou um desempenho superior com a mesma qualidade. Associado com o baixo custo de componentes de FPGA (menor que uma placa Vídeo for Windows), o sistema resultante é um dispositivo de pequeno tamanho que serve bem para robôs autônomos. A avaliação das imagens resultante permitiu concluir que a qualidade dos resultados obtidos também foi satisfatória.

Através dos resultados obtidos pode-se concluir que a implementação em Hardware Programável torna o sistema extremamente eficiente e que essa eficiência não pode ser aproveitada plenamente pelo conjunto do robô, pois os componentes restantes

não conseguem alcançar o mesmo nível de desempenho.

Como trabalho futuro em VHDL deseja-se implementar outros algoritmos de Visão Computacional em VHDL, como por exemplo, o *Blob Colouring* (Ballard & Brown, 1982). Finalmente, deseja-se utilizar os algoritmos implementados em FPGAs no protótipo de robô para um time da liga *Small Size* da *RoboCup* construído na FEI (mostrado na figura 11).

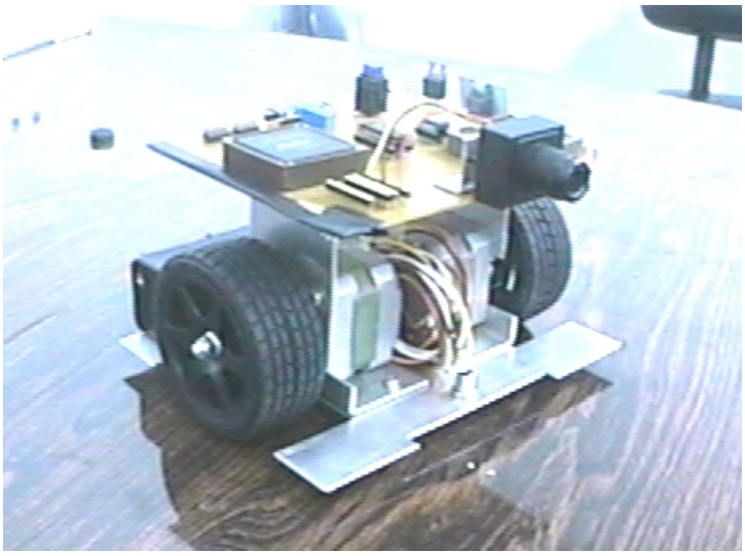


Figura 11 – Protótipo do Robô Autônomo para a RoboCup F180 construído na FEI.

REFERÊNCIAS BIBLIOGRÁFICAS

ALOIMONOS, Y. What I have learned. **CVGIP: Image Understanding**, v.60, n.1, p.74-85, July 1994.

BALLARD, D. H.; BROWN, C. M. **Computer Vision.** Englewood Cliffs, Prentice-Hall, 1982.

BHASKER, J.: **A VHDL Primer**. Prentice Hall, Englewood Cliffs, 1995.

BROWN, S.; ROSE, J. Architecture of FPGAs and CPLDs: A Tutorial. **IEEE Design and Test of Computers**, v. 13, n. 2 , p.42-57, 1996.

KITANO, H. et al. "RoboCup: A challenge Problem for AI". **AI Magazine**, v. 18, n. 1, p. 73-85, Spring 1997.

KITCHIN, P. W.; PUGH, A. Processing of Binary Images. In: PUGH, A. (editor) **Robot Vision.** Berlin, Springer, 1983. P. 21–42.

KRAETZCHMAR, G. et al. The ULM Sparrows: Research into Sensorimotor Integration, Agency, Learning, and Multiagent Cooperation. In: ROBOCUP WORKSHOP, 2, Paris, 1998. **Proceedings.** FIRA, 1998. p. 459- 465

LI, Z. N. **Multimedia Systems Class Notes**. - School of Computing Science - Simon Fraser University, 1998. (http://www.cs.sfu.ca/fas-info/cs/CC/365/li/material/notes/Chap3/Chap3.3/Big.html)

RILLO, A.H.R.C. **Sistema de Visão Binária com Reconhecimento de Peças Parcialmente Oclusas.** São Bernardo do Campo, 1989. Dissertação (Mestrado) - Faculdade de Engenharia Industrial.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach.** Englewood Cliffs, Prentice Hall, 1995.

SANDERSON, A. "Micro-Robot World Cup Soccer Tournament (MiroSot)". **IEEE Robotics and Automation Magazine**, pg.15, December 1997.